LINGI1341 : P1 - Rapport

Dewit Alexandre
Yakoub Jacque

UCL

Ce projet est à faire par groupe de deux. Le but du projet est d'implémenter en langage C un protocole de transport utilisant des segments UDP. Ce protocole permettra de réaliser des transferts fiables de fichiers, utilisera la stratégie du selective repeat et permettra la troncation de payload. Il devra également fonctionner avec IPv6. Deux programmes sont à réaliser, permettant de faire un transfert de données entre deux machines distantes.

Table des matières

Introduction

Ce rapport porte sur nos choix d'implémentation lors de la conception du programme. Nous parlerons donc de la gestion du timestamp, de la réception des paquets **PTYPE_NACK**, du retransmission timeout et de la partie critique de notre programme.

1 Le timestamp

Nous mettons dans ce champ le temps en ms depuis le premier janvier 1970. Le timestamp sert dans notre cas à calculer le RTT. Cependant, le temps de réponse étant tellement petit entre le serveur et le client, nous pourrions avoir des résultats incohérents car les valeurs sont tributaires du paramétrage de l'horloge sur le système concerné. C'est donc ainsi que la solution *gettimeofday()* a offert une solution supposée multiplateforme.

2 La gestion des paquets PTYPE_NACK

Le client recevant ce type de paquet, cela voudrait dire que le réseau est congestionné et que ça ne sert à rien de renvoyer des paquets. A cet effet, on décrémente la taille de la window du client. Lors d'un acquittement, nous incrémentons ce compteur en fonction du nombre de paquet correctement envoyés.

3 Le retransmission timeout

Au tout début, le client envoie un premier paquet tronqué (champ tr = 1) avec un numéro de séquence valide et un champ length à 0. Le serveur qui recevra ce paquet va réagir en créant un paquet **PTYPE_NACK**. Dès réception de ce paquet, le client va pouvoir estimer le **RTT**.

Pour ce faire, on utilise 2 fois la méthode *gettimeofday()* :

1. Juste avant d'envoyer le premier paquet au serveur
2. Juste après avoir récupéré le paquet du serveur en réponse

En faisant la différence de ces deux temps, nous pouvons aisément calculer le RTT. Par la suite, ce même procédé est appliqué pour les autres paquets, le résultat étant la moyenne des RTT.

$RTT_{moyen} = \frac{RTT_2 - RTT_1}{2}$

4 La partie critique du programme

4.1 Choix du timer est important

Un timer trop petit , bien qu'efficace quand il n'y a pas de perte/réordonnancement, donnera lieu à beaucoup de pertes (les paquets devront être retransmis très rapidement) tandis qu'un timer long ne sera pas optimisé (on attend trop de temps).

4.2 La taille de la window du serveur

Si la taille de la window coté serveur est trop grande, on risque d'avoir un soucis car au final on va stocker beaucoup trop de paquets. Afin de l'expérimenter, la taille maximale de la window coté serveur est de 31 slots.

4.3 La taille de la window coté client

Si la taille de la window coté client est trop grande, alors on risque d'avoir des problèmes de performance (ex : retransmettre plusieurs fois le même paquet si les paquets ne sont pas envoyés dans l'ordre attendu). Nous avons donc choisi une taille maximale de 5 slots.

5 Notre stratégie de tests

Il est assez difficile de tester le fonctionnement du serveur et du client. Nous avons donc opté pour des tests visant à vérifier le fonctionnement des méthodes traitant les paquets (getter, setter, encode, decode) et la window coté serveur (ex : stocker un paquet si le numéro de séquence de celui ci correspond aux numéros de séquences valides de la window serveur).

6 Architecture du programme

Nous allons parcourir les différents sous-dossiers sources de notre projet afin d'expliquer leur utilisation.

6.1 client

client C'est la main du client qui fait toutes les opérations.

clientUtil Méthodes génériques pour gérer la window coté client.

receivedACKorNACK Permet de gérer les cas spécifiques tels que le ACK ou le NACK coté client.

resendLostMessages Permet de renvoyer des paquets non réceptionnés

sendMessage Permet de lire sur STDIN et de renvoyer un message

estimateRTTAndWindowSize Permet de gérer l'envoi du premier paquet permettant de calculer une estimation du RTT.

6.2 common

commandLine Permet de gérer la réception d'arguments en ligne de commande.

6.3 packet_table

packet_table Permet de gérer une table de pointeurs vers des paquets (add, remove et get).

6.4 paquet

packet_implem Permet de gérer un paquet (getter, setter, encode, decode).

6.5 SendAndReceiveData

connexionHelper Regroupe toutes les opérations permettant de créer un socket, lire un socket et écrire sur stdout, convertir un nom de domaine ou une adresse IPV6 en structure lisible pour la machine.

6.6 server

server C'est la main du serveur qui fait toutes les opérations.

6.7 server_window

server_window_util Ce fichier contient toutes les méthodes permettant de gérer la window coté serveur. On crée une nouvelle structure de stocker différentes informations comme le dernier numéro de séquence valide reçu, la liste des paquets stockés, la taille de la window du serveur/client et la liste des numéros de séquences déjà reçus.

7 Conclusion

Nous voici arrivé à la fin de cette intense mission d'implémentation, celle-ci nous aura permis de comprendre de manière plus pratique la réalisation d'un protocole de transfert.
Nous avons effectué quelques tests à l'aide du simulateur de lien [1] et du script de test présent sur Moodle afin d'expérimenter des situations potentielles. Nous avons retenu ce scénario dans lequel se produisent plusieurs situations typiques de ce prototole.

```
Use IPV6 Address: ::1
Socket successfully created - listenning to port 1341
Trying to send a false packed to get RTT
Trying to send a false packed to get RTT
Trying to send a false packed to get RTT
Initial calculated RTT : 100 ms
Initial window size of receiver : 31
A new payload is ready to be read from STDIN
        Setting the packet N° 0 to be send
        Payload : 512
        Packet correctly encoded - ready to be sent
        Packet correctly sent
~
~
```

Logs de la partie sender

```
Socket successfully bind - listening to port 2456
Waiting for client
A client is connected
Received message of 12 bytes
         Is valid ? : 1
        Received valid PACKET N° 1
        This packet was truncated
Received message of 528 bytes
        Could not decode the packet - errcode : 3
         Is valid ? : 0
~
~
```

Logs de la partie receiver

Nous pouvons lister ces cas présentés ci-dessus :

- La perte d'un paquet (le paquet tronqué pour connaitre le RTT)
- L'encodage d'un message, tout en préparé à recevoir des données
- La réception et la gestion d'un packet tronqué ou corrompu (potentiellement modifié par le réseau)

Enfin, nous avons fait notre possible afin de ne pas laisser de fuite de mémoire critiques dans les diverses parties de notre code en utilisant notamment Valgrind (ou gdb dans certains cas).

Nous comparerons notre version du projet avec un autre binome lors de la séance d'interopérabilité afin de constater les éventuelles différences entre nos versions et d'adapter en conséquence.

1. Source : `https://github.com/oliviertilmans/LINGI1341-linksim`